# Week 5 Assignment – Hospital Management System

## Hospital Management System

The **Hospital Management System** uses Binary Trees, Threaded. Binary Trees and Binary Search Tree to manage the hospital’s department hierarchy and patient records. The system supports the following features:

* Represent the hierarchical structure of the hospital’s department
* Manage patient records within each department, allowing efficient search and manipulation
* Provide efficient in-order traversal of patient records for reporting
* Patients can be added, updated or removed dynamically
* Display department and patient records

1. Manage Departments
   1. Represent the hierarchy of the departments eg: Cardiology, Surgery, Neurology, Urology, etc
   2. Each department has a name and can have sub-departments
   3. Add sub-department to a department and display all departments in hierarchical manner
2. Manage patient records
   1. Each department to store patient data sorted by patientId.
      1. Each patient record includes PatientId (unique int key) and patient name
   2. The system should allow to add new patient records, delete patient record by patientId and display all patient records in ascending order of PatientId’s
3. Efficient generation of patient reports
   1. Store patient records for specific department
   2. Display all patient records using non-recursive in-order traversal
4. Integration
   1. Combine all tree structures to manage departments and patient records
   2. Assign departments to their respective patient records

Solution:

// Online C++ compiler to run C++ program online

#include <iostream>

#include <string>

using namespace std;

// Abstract Base Class

class Tree

{

public:

virtual void add(int id, string name, int age, string illness = "") = 0;

virtual void update(int id, string name, int age, string illness = "") = 0;

virtual void remove(int id) = 0;

virtual bool search(int id) = 0;

virtual void display() = 0;

virtual ~Tree() {}

};

// Patient Node for Binary Search Tree (BST)

struct PatientNode

{

int patientID;

string name;

int age;

string illness;

PatientNode\* left;

PatientNode\* right;

PatientNode(int id, string n, int a, string ill)

: patientID(id), name(n), age(a), illness(ill), left(nullptr), right(nullptr) {}

};

// Patient Class (BST Implementation)

class Patient : public Tree

{

private:

PatientNode\* root;

// Helper Functions

void insertPatient(PatientNode\*& root, int id, string name, int age, string illness)

{

if (root == nullptr)

{

root = new PatientNode(id, name, age, illness);

return;

}

if (id < root->patientID)

insertPatient(root->left, id, name, age, illness);

else if (id > root->patientID)

insertPatient(root->right, id, name, age, illness);

else

cout << "Patient with ID " << id << " already exists"<<endl;

}

PatientNode\* searchPatient(PatientNode\* root, int id)

{

if (root == nullptr) return nullptr;

if (id == root->patientID) return root;

if (id < root->patientID)

return searchPatient(root->left, id);

return searchPatient(root->right, id);

}

PatientNode\* deletePatient(PatientNode\*& root, int id)

{

if (root == nullptr) return nullptr;

if (id < root->patientID)

root->left = deletePatient(root->left, id);

else if (id > root->patientID)

root->right = deletePatient(root->right, id);

else

{

if (root->left == nullptr)

{

PatientNode\* temp = root->right;

delete root;

return temp;

}

else if (root->right == nullptr)

{

PatientNode\* temp = root->left;

delete root;

return temp;

}

PatientNode\* successor = findMin(root->right);

root->patientID = successor->patientID;

root->name = successor->name;

root->age = successor->age;

root->illness = successor->illness;

root->right = deletePatient(root->right, successor->patientID);

}

return root;

}

PatientNode\* findMin(PatientNode\* node)

{

while (node && node->left)

node = node->left;

return node;

}

void inorder(PatientNode\* root)

{

if (root == nullptr) return;

inorder(root->left);

cout << "ID: " << root->patientID << ", Name: " << root->name << ", Age: " << root->age << ", Illness: " << root->illness << endl;

inorder(root->right);

}

public:

Patient() : root(nullptr) {}

void add(int id, string name, int age, string illness = "") override

{

insertPatient(root, id, name, age, illness);

}

void update(int id, string name, int age, string illness = "") override

{

PatientNode\* patient = searchPatient(root, id);

if (patient)

{

patient->name = name;

patient->age = age;

patient->illness = illness;

cout << "Patient with ID " << id << " updated successfully"<<endl;

}

else

{

cout << "Patient with ID " << id << " not found.\n";

}

}

void remove(int id) override

{

if (searchPatient(root, id))

{

root = deletePatient(root, id);

cout << "Patient with ID " << id << " deleted successfully.\n";

}

else

{

cout << "Patient with ID " << id << " not found.\n";

}

}

bool search(int id) override

{

return searchPatient(root, id) != nullptr;

}

void display() override

{

inorder(root);

}

};

// Department Node for Threaded Binary Tree (TBST)

struct DepartmentNode

{

string name;

Patient patients; // Each department manages its own patients

DepartmentNode\* left;

DepartmentNode\* right;

bool isThreaded;

DepartmentNode(string deptName)

: name(deptName), left(nullptr), right(nullptr), isThreaded(false) {}

};

// Department Class (TBST Implementation)

class Department : public Tree

{

private:

DepartmentNode\* root;

DepartmentNode\* addDepartment(DepartmentNode\* root, string name)

{

if (root == nullptr)

return new DepartmentNode(name);

DepartmentNode\* parent = nullptr;

DepartmentNode\* current = root;

while (current != nullptr)

{

parent = current;

if (name < current->name)

{

if (current->left == nullptr)

break;

current = current->left;

}

else

{

if (current->isThreaded || current->right == nullptr)

break;

current = current->right;

}

}

DepartmentNode\* newNode = new DepartmentNode(name);

if (name < parent->name)

parent->left = newNode;

else

{

newNode->right = parent->right;

parent->right = newNode;

parent->isThreaded = false;

}

return root;

}

DepartmentNode\* searchDepartment(DepartmentNode\* root, string name)

{

while (root != nullptr)

{

if (name == root->name)

return root;

if (name < root->name)

root = root->left;

else if (root->isThreaded)

break;

else

root = root->right;

}

return nullptr;

}

void inorderDepartments(DepartmentNode\* root)

{

DepartmentNode\* current = root;

while (current != nullptr && current->left != nullptr)

current = current->left;

while (current != nullptr)

{

cout << "Department: " << current->name << endl;

current->patients.display();

if (current->isThreaded)

current = current->right;

else

{

current = current->right;

while (current != nullptr && current->left != nullptr)

current = current->left;

}

}

}

public:

Department() : root(nullptr) {}

void add(int id, string name, int age, string illness = "") override

{

cout << "This operation is not applicable at the department level"<<endl;

}

void update(int id, string name, int age, string illness = "") override

{

cout << "This operation is not applicable at the department level"<<endl;

}

void remove(int id) override

{

cout << "This operation is not applicable at the department level"<<endl;

}

bool search(int id) override

{

return false;

}

void display() override

{

inorderDepartments(root);

}

void addDepartment(string name)

{

root = addDepartment(root, name);

}

void addPatientToDepartment(string deptName, int id, string name, int age, string illness)

{

DepartmentNode\* dept = searchDepartment(root, deptName);

if (dept)

dept->patients.add(id, name, age, illness);

else

cout << "Department not found."<<endl;

}

void removePatientFromDepartment(string deptName, int id)

{

DepartmentNode\* dept = searchDepartment(root, deptName);

if (dept)

{

if (dept->patients.search(id))

{

dept->patients.remove(id);

}

else

{

cout << "Patient with ID " << id << " not found in " << deptName << " department"<<endl;

}

}

else

{

cout << "Department not found"<<endl;

}

}

};

// Main Function

int main()

{

Department dm;

// Add Departments

dm.addDepartment("Cardiology");

dm.addDepartment("Orthopedics");

dm.addDepartment("Neurology");

// Add Patients

dm.addPatientToDepartment("Cardiology", 101, "CP1", 30, "Heart Issue");

dm.addPatientToDepartment("Cardiology", 100, "CP2", 20, "Heart Issue");

dm.addPatientToDepartment("Orthopedics", 102, "OP1", 40, "Fracture");

dm.addPatientToDepartment("Neurology", 103, "NP1", 50, "Migraine");

dm.display();

// Remove a Patient

dm.removePatientFromDepartment("Cardiology", 101);

// Display Departments and Patients

dm.display();

return 0;

}